الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

N° Réf :……

Centre Universitaire
Abd elhafid boussouf Mila
Institut des Sciences et de la Technologie Département de Mathématiques et Informatique

Mémoire préparé En vue de l'obtention du diplôme de Licence

En : - Filière : Informatique générale

Comparaison entre les algorithmes d'équilibrage de charge dynamique « sender_initiated » et « resever_initiated »

Préparé par :

- **Boussouf Tarek**
- **Kadja Mouad**
- **Boussouf Mohammed Amin**

Encadrer par : Ali Widad

Année universitaire : 2014/2015

Remerciement

Nous remercions le bon DIEU qui nous a donné la vie et la santé pour pouvoir réaliser ce mémoire et nous prions pour son prophète MOUHAMED (psl).

En préambule à ce mémoire, nous souhaitons adresser nos remerciements les plus sincères aux personnes qui nous ont apporté leur aide et qui ont contribué à l'élaboration de ce modeste travail ainsi qu'à la réussite de cette formidable formation.

Nous remercions a notre encadreur Mme ali widad d'avoir nous a encadré tout au long de l'étude, pour sa disponibilité, ses conseils, ses suggestions ainsi que le temps qu'il nous a consacré.

Nous exprimons notre gratitude et notre reconnaissance à nos parents pour leurs contributions, leurs soutiens et leurs patiences.

Nous tenons à exprimer nos sincères remerciements à tous les personnels de l'institut de sciences et de la technologie du centre universitaire de Mila surtout les enseignants qui nous ont enseigné durant toutes nos années d'étude.

Nous remercions les étudiants de la promotion 2014/2015 pour avoir été liés et unis tout au long de cette année et tous ceux qui ont collaborés de prés ou de loin à l'élaboration de ce travail. Qu'ils acceptent nos humbles remerciements

BOUSSOUF MOHAMED AMIN BOUSSOUF TAREK KADJA MOUAD

Dédicaces

Nous dédions ce travail

A nos parents qui sont toujours là pour nous.

A nos frères

Et toute nos famille boussouf et kaja

A tout nos amies

A tous les étudiants de 3ème année informatique

A tous mes enseignants

A tous ceux qui me connaissent

Résumé :

L’équilibrage de charge ont des algorithmes peuvent à résoudre des problèmes des charge entre les poste sous chargé et surchargé, cela pour améliorer les fonctionnements des travaux ainsi que des connaissances les mieux solutions et méthodes pour réaliser les problèmes.

Dans le cadre de la résolution de ce problème, nous proposons dans ce mémoire notre contribution et nos analyses. Cette contribution consiste dans propose un algorithme d’équilibrage de charge, dont les principaux objectifs sont la réduction du temps de réponse et du coût de transfert des tâches soumises.

Abstract: (en englais)

Load balancing have algorithms can solve the problems of load between post in charge and responsible, that to improve the functioning of labor as well as the best solutions and methods to achieve the knowledge problems.

In order to solve this problem, we present in this report our contribution and Analyses.

This contribution consists in developing algorithms for load balancing, whose main objectives are to reduce response time and communication cost of the submitted jobs.

Table de matière

Table de figure

Les tableaux

Algorithme

Introduction général:

De nos jours, le progrès s'observe dans de nombreux domaines ce qui résulte aujourd'hui d'un large réseau informatique. Cette évolution a alors entraîné dans un premier temps une pression énorme sur le niveau du matériel informatique. Par conséquent, cette évolution a conduit au déséquilibre de son fonctionnement et a alors entraîné l'émergence d'un certain nombre de problèmes qui entravent le processus de l'emploi. Quels modes et mécanismes, les informaticiens peuvent trouver pour chercher des solutions à ce problème?

Nous avons trouvé des solutions qui ont joué un rôle efficace dans l'élimination de ces problèmes. Parmi ces problèmes, il y a le problème d'équilibrage de charge entre les postes sous chargés et surchargés. L'équilibre entre les fonctions de ces postes est une solution à ce problème. Ce dernier suit des étapes pour atteindre cet objectif. Il y a alors des algorithmes entre l'émetteur et le récepteur.

Dans notre premier chapitre consacré à l'état de l'art, nous allons identifier l'équilibrage de charge, ces politiques, ainsi que ces algorithmes et les stratégies source et serveur initiative. De ce fait, nous avons appris des éléments comme l'algorithme de Sender, l'algorithme de receveur et l'algorithme de Sender/receveur.

Dans le deuxième chapitre, nous allons parler de conception, c'est à dire que nous allons expliquer les algorithmes d'équilibrage de charge dans le but de simplifier mes travaux de ce problème.

Enfin, dans le dernier chapitre de l'implémentation, nous allons parler de langage « C ». Nous expliquerons la surface de ces algorithmes, qui va notamment aider à la comparaison entre les algorithmes et ainsi, ils vont nous permettre d'identifier la solution optimale pour le problème d'équilibrage.

Tout cela afin que nous puissions maintenir le travail réussi, gagner du temps et ainsi assurer la sécurité du matériel.

Chapitre 1 Etat de l'art

1 Introduction

Le progrès technologique réalisé dans la conception des processeurs a permis d'offrir aux

Utilisateurs des machines ayant des puissances de calcul considérables. Généralement, les utilisateurs utilisent seulement une fraction de la capacité de traitement de leurs machines.

Ces machines se trouvent alors inutilisées pendant de longues durées. Des études ont montré que les stations de travail sont inutilisées jusqu'à75% du temps. Selon une étude du Los Alamos National Lab/USA, les stations de travail ne sont en service que pendant moins de 10% du temps.

Néanmoins, il y a des périodes où un utilisateur désire exécuter des tâches qui demandent une puissance de calcul supérieure à la capacité effective de sa machine. Il se voit alors obligé d'attendre pendant parfois de longues durées avant d'obtenir les ressources requises.

Donc, un problème important à résoudre est l'allocation des ressources CPU dans le but de répartir équitablement l'ensemble du travail sur les nœuds du système. Contrairement au partage de charge (load sharing), qui consiste uniquement à répartir les processus (ou tâches) entre les nœuds libres, l'équilibrage de charge (load balancing) consiste à équilibrer la charge entre l'ensemble des nœuds. Les deux termes, équilibrage de charge et partage de charge, sont souvent utilisés dans la littérature de façon interchangeable. Nous utilisons tout au long de ce chapitre le terme distribution de charge pour designer aussi bien l'équilibrage de charge que le partage de charge. Les objectifs les plus usuels de la distribution de charge incluent la minimisation du temps de réponse moyen des tâches, la maximisation du débit moyen du système, la distribution équilibrée de la charge, la minimisation du temps d'inoccupation des processeurs, et l'augmentation de la fiabilité du système (tolérance aux fautes).

2 L'équilibrage de charge

L'équilibrage de charge est le processus de distribuer la charge d'une application parallèle sur un ensemble de processeurs pour améliorer l'exécution en réduisant le temps de réponse de l'application. Les décisions de quand, où et quelle charge doivent être transférées sont critiques, et donc l'information de charge doit être précise et à jour [1].

Si la décision d'équilibrage de charge est faite avant l'exécution de l'application et en connaissant de toutes les variables qui peuvent affecter cette exécution, on parle d'un équilibrage de charge statique. Mais, si on ne connaît pas toutes les variables qui peuvent affecter l'exécution et donc les décisions d'équilibrage doivent être faites pendant l'exécution, on parle d'un équilibrage de charge dynamique. Dans l'équilibrage de charge dynamique, les décisions dépendent de l'information rassemblée du système. L'information de charge peut être mise en commun entre les processeurs périodiquement ou sur la demande, avec des collecteurs centralisés ou distribués de l'information [2].Le travail de Casavant et de Kuhl [3] reporte qu'un temps de réponse plus rapide est plus important que la stabilité pour améliorer l'exécution. Notre recherche se focalise dans l'équilibrage de charge dynamique.

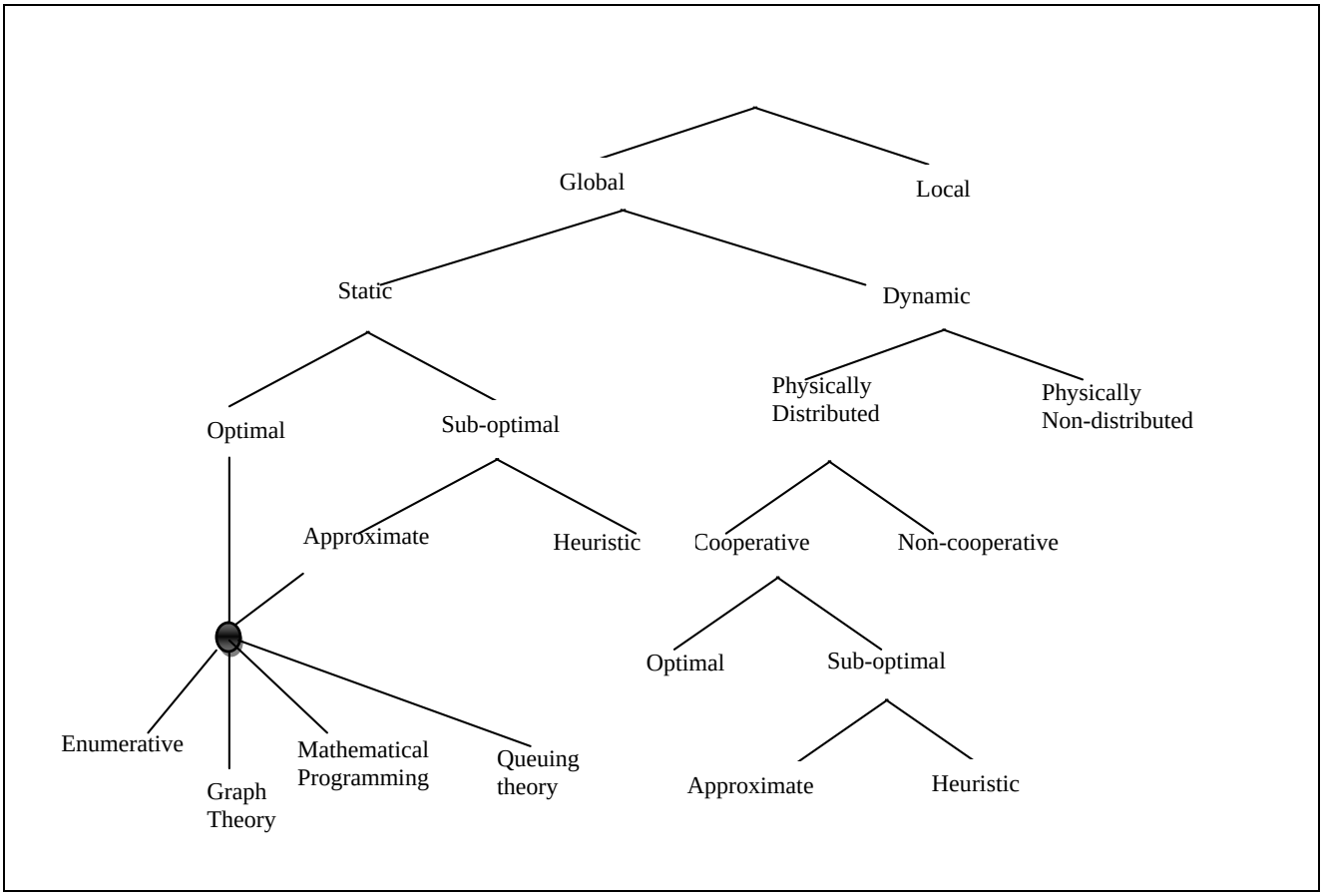

Figure.1 : Une taxonomie de Casavant and Kuhl dans les systèmes

Le problème de l'équilibrage étant un problème relativement ancien, beaucoup d'approches ont été proposées pour le résoudre. Casavant et Kuhl [4] ont défini une taxonomie largement adoptée par la communauté scientifique dont les principales classes sont :

1. Approche statique Vs. approche dynamique Dans une approche statique, les tâches sont assignées aux machines avant l'exécution de l'application qui les contient. Les informations concernant le temps d'exécution des tâches et les caractéristiques dynamiques des machines sont supposées connues a priori. Cette approche est efficace et simple à mettre en œuvre lorsque la charge de travail est au préalable suffisamment bien caractérisée. Dans une approche dynamique, l'assignation des tâches aux machines se décide durant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ceci permet d'améliorer les performances d'exécution des tâches mais au prix d'une complexité dans la mise en œuvre de cette stratégie, notamment en ce qui concerne la définition de l'état de charge du système, qui doit se faire de manière continue.

2. Approche centralisée Vs. approche distribuée Dans une approche centralisée, un site du système est choisi comme coordinateur. Il reçoit les informations de charge de tous les autres sites qu'il assemble pour obtenir l'état de charge global du système. Dans le cas d'une approche distribuée, chaque site du système est responsable de collecter les informations de charge sur les autres sites et de les rassembler pour obtenir l'état global du système. Les décisions de placement de tâches sont prises localement, étant donné que tous les sites ont la même perception de la charge globale du système.

3. Approche source-initiative Vs. receveur-initiative L'approche source-initiative est appliquée lorsqu'un site, appelé source, détecte qu'il a une surcharge de travail et qu'il cherche à transférer le surplus vers un site faiblement chargé. L'approche receveur- initiative s'applique lorsqu'un site faiblement chargé, appelé receveur, demande à recevoir tout ou partie du surplus des sites surchargés.

2.1 Équilibrage de charge statique

Dans une approche statique, les tâches sont assignées aux machines avant l'exécution de l'application qui les contient. Les informations concernant le temps d'exécution des tâches et les caractéristiques dynamiques des machines sont supposées connues a priori. Cette approche est efficace et simple à mettre en œuvre lorsque la charge de travail est au préalable suffisamment bien caractérisée.

Le problème de l'équilibrage étant un problème relativement ancien, beaucoup d'approches ont été proposées pour le résoudre. Casavant et Kuhl [4] ont défini une taxonomie d'ordonnancement largement adoptée par la communauté scientifique, ils ont proposé des catégories pour les algorithmes de distribution de tâche statique :

1. **Programmation mathématique :** quant à elles, se ramène à un problème d'optimisation combinatoire. Elles utilisent en général les techniques de la programmation non linéaire en nombres entiers pour sa résolution. On utilise alors 0des techniques de programmation mathématique comme par exemple la programmation dynamique [5,6] et les procédures par séparation et évaluation (Branch and Bound) [7, 8]. En général, il s'agit de minimiser les fonctions des coûts, tout en respectant un certain nombre de contraintes. On peut s'intéresser aux contraintes temporelles, de charge, de ressources, de redondance.

2. **La théorie des graphes** : basées sur la théorie des graphes modélisent le problème par des graphes et utilisent des techniques de traitement de graphes pour le résoudre, en utilisant le partitionnement de graphe pour réduire au minimum l'exécution, la communication et des coûts de réaffectation [9], Ou, ayant le modèle d'interconnexion des tâches en forme d'arbre, un algorithme réduit au minimum la somme d'exécution et les coûts de communication pour systèmes distribués arbitrairement connectés avec nombres arbitraires de Processeurs trouvant le minimal arbre couvrant(spanning tree).

2.2 Équilibrage de charge dynamique

Dans une approche dynamique, l'assignation des tâches aux machines se décide durant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ceci permet d'améliorer les performances d'exécution des tâches mais au prix d'une complexité dans la mise en œuvre de cette stratégie, notamment en ce qui concerne la définition de l'état de charge du système, qui doit se faire de manière continue.

La conception d'une stratégie d'équilibrage est directement liée à l'objectif de la distribution. Riedl et Richter présentent dans leur travail une liste d'objectifs principaux [10] :

- Performance de service pour tâches : temps d'attente, temps de service, temps de réponse, disponibilité de services.
- distance Physique entre tâches et données
- performance de Service de ressources : débit, cache ou temps de communication.
- égalisation de charge entre processeurs.
- Minimisation de temps d'inoccupé de processeur.

À partir de ces objectifs, une métrique (unité de mesure pour déterminer les déséquilibres de charge) peut être choisie pour déterminer la performance du système.

En considérant les objectifs, nous pouvons étudier la performance de deux perspectives : celui du système et celui de l'application parallèle. Pour le point de vue d'application parallèle, généralement la métrique utilisée est le temps d'achèvement du processus individuel (aussi connu comme makespan). Et,

du point de vue du système, communément la métrique utilisée vise à une maximisation (ou plutôt la distribution) d'utilisation des ressources. Un compromis peut être atteint en choisissant la métrique qui incorpore à la fois les deux points de vue, parce que les applications vont essayer d'utiliser les ressources disponibles pour améliorer leur performance et les systèmes viseront la distribution juste des ressources.

Le travail de Casavant et Kuhl [3] proposent en plus deux propriétés pour considération dans l'évaluation d'un mécanisme de distribution de charge :

- Performance : la mesure quantitative de l'amélioration de l'application parallèle quand le mécanisme gère les ressources.
- Efficacité : les coûts produits par le gestionnaire de ressource.

Un parfait algorithme d'équilibrage de charge est celui qui réalise la meilleure performance possible avec le coût minimal.

Thomas Kunz a décrit dans [11] quelques exigences qui se sont avérées être importantes pour un objectif général d'une stratégie d'équilibrage de charge :

1 Aucune connaissance a priori d'exigence de tâche entrante

2. Aucune hypothèses sur le réseau sous-jacent (topologie, homogénéité, taille, etc)

3. Dynamique, physiquement distribuée et prise de décisions coopérative

4. Minimisation de temps de réponse moyen/pire de tâches comme critères de performance. Définir le temps de réponse comme le temps entre une tâche est reçue par une application parallèle et elle est terminé par le processeur.

Néanmoins, Kunz [11] a conduit son étude en 1991 avec des réseaux hétérogènes sans varier la distance géographique. Aujourd'hui, avec l'utilisation d'Internet et réseaux à grande échelle pour le calcul parallèle, la deuxième exigence proposée par Kunz a devient inapplicable.une connaissance dans le réseau sous-jacent a la haute importance dans des algorithmes d'équilibrage de charge modernes. Une autre condition importante dans des algorithmes d'équilibrage de charge est leur niveau de complexité. Mirchandaney, Towsley et Stankovic ont rapporté dans leur travail [12] que :

La distribution de charge simple produit amélioration de performance dramatique comparée à une configuration sans équilibrage de charge.

Politiques complexes, qu'essayent de faire la meilleure sélection, n'offrent pas de nouvelles améliorations.

3 Taxonomie des approches d'équilibrage de charge

Les recherches menées dans le développement des approches d'équilibrage de charge, ont permis de distinguer plusieurs catégories d'approche. Elles peuvent être: statiques ou dynamiques, locales ou globales, centralisées ou distribuées, coopératives ou non coopératives, source-initiative, receveur initiative ou symétrique-initiative.

3.1 Equilibrage de charge statique Vs dynamique

Affecte les taches à des ressources fixes, sans prendre en considération l'état du système ou les changements pouvant y survenir. Comme exemple de stratégie statique, nous pouvons citer la méthode du round-robin, qui implique approximativement, l'exécution par une ressource du même nombre de requêtes. Cette approche est bénéfique si les caractéristiques de la charge de travail sont connues et si l'état du système ne change pas fréquemment. La distribution dynamique [14] permet d'outre passer les limites de la distribution statique, a savoir la non connaissance des caractéristiques de la charge de travail et la variation des performances des ressources. On peut distinguer plusieurs interprétations de l'équilibrage de charge dynamique [15]. Il peut ainsi designer soit la variation de la charge pendant le processus de l'équilibrage de charge, soit la variation de la topologie de la plate-forme (réseau (dynamique) ayant des liens de communications dynamiques (non fiables). Différentes approches sont utilisées pour traiter l'équilibrage de charge dynamique, dont nous pouvons citer:

- Utiliser le passe pour prédire l'avenir. Autrement dit, utiliser la vitesse de calcul observée pour chaque ressource afin de décider de la redistribution du travail.

- Redistribuer les données parmi les ressources participantes pendant l'exécution de l'algorithme ; cette redistribution est réalisée par le transfert de données des ressources les plus chargées vers les moins chargées.

Cette phase d'équilibrage de charge peut être centralisée par une seule ressource ou être distribuée sur l'ensemble des ressources.

- Toutes les ressources informent leurs voisins proches de leur niveau de charge et mettent a jour ces informations pendant l'exécution de l'application. Ainsi, en définissant un certain seuil (seuil qui permet de définir la surcharge ou sous-charge d'une ressource), les ressources surchargées peuvent envoyer des données aux ressources sous-chargées.

3.2 Equilibrage de charge local Vs global

Dans l'équilibrage de charge local, chaque ressource collecte les Informations de ses voisins pour décider du transfert de charge et réaliser un équilibrage local. L'intérêt majeur est de minimiser les Communications distantes et réaliser efficacement l'équilibrage de la ressource.

Cependant, dans l'équilibrage global, l'information globale du système est nécessaire pour initier l'équilibrage de charge. Cette stratégie requiert un échange important d'informations dans le système, ce qui peut affecter ses performances.

3.3 Equilibrage de charge centralise Vs distribue

Dans l'approche centralisée la collecte et la gestion de l'information de charge sont effectuées par une seule ressource désignée comme coordinateur. Elle reçoit l'information courante de toutes les autres ressources et l'assemble dans un vecteur de charge. Lorsqu'une ressource décide de transférer une tache, elle envoie une demande au coordinateur qui choisit alors une ressource cible, en utilisant le vecteur de charge, et informe la ressource source de ce choix [16].

L'avantage de cette approche est qu'elle réduit les surcouts du système grâce à la centralisation de l'information de charge. Cependant, on peut distinguer les inconvénients suivants :

- Cette méthode peut engendrer un décalage de temps important, compte tenu des délais de transfert et de la perte de messages.

- Le coordinateur peut être l'objet d'un goulot d'étranglement limitant la distribution des messages depuis et vers cette ressource.

- Une panne au niveau du coordinateur provoquera l'effondrement du système.

Pour remédier a cette situation, des mécanismes redondants (de réplication) sont souvent utilises pour remplacer les taches défaillantes.

L'approche distribuée permet de remédier au problème de goulot d'étranglement, constate dans l'approche précédente, lors de la collecte de l'information de charge. Ceci est réalise en distribuant la gestion de la prise de décision, ou chaque ressource construit de manière autonome son propre vecteur de charge en rassemblant l'information de charge des n'autres ressources.

Les décisions de placement sont faites localement en utilisant les vecteurs de charge locaux. Ceci permet, dans le cas ou un coordinateur tombe en panne, lors de l'exécution d'une tache, de continuer la prise de décisions par les autres taches Grace a cette répartition, il est possible de maintenir, d'ajouter ou de retirer une ressource défectueuse, et d'un autre cote d'accélérer de manière significative le processus de prise de décision. Cependant, la gestion est plus compliquée et le cout des communications engendrées peut être élève.

Une autre approche mixte combine les deux modèles précédents et permet ainsi, de profiter des avantages de chacun d'eux. Dans ce cas, le coordinateur établit de manière centralisée son vecteur de charge et le distribue périodiquement aux autres ressources qui prennent localement leurs décisions de placement .

3.4 Source/receveur/symétrique initiative d'équilibrage

Les techniques d'ordonnancement des taches dans les systèmes distribuent

Sont principalement divisées en source-initiative, receveur-initiative et symétrique-initiative.

Source-initiative et receveur-initiative ont été présentes dans les problèmes. Symétrique-initiative combine les deux stratégies précédentes et permet donc d'initier le transfert de charge soit par les ressources surchargées ou celles légèrement chargées [17].

Coopératif Vs non-coopératif dans un système distribue coopératif, plusieurs ressources participent aux prises de décisions en coopérant. Elles travaillent vers un but commun tout en gardant leur autonomie propre. Le concept de base est la négociation et la soumission de propositions par la ressource demanderesse. Par contre, dans un système non-coopératif, les ressources prennent des décisions individuellement. Elles agissent comme des entités autonomes orientées vers un but individuel, leurs décisions n'affectent que leurs performances locales.

Toutefois, elles Peuvent avoir des effets secondaires sur le fonctionnement global du system .

4 Les composants d'algorithme d'équilibrage de charge

L'´equilibrage de charge est le processus de distribuer la charge d'une application parallèle sur un ensemble de processeurs pour améliorer l'exécution en réduisant le temps de réponse de l'application. Les décisions de quand, ou et quelles charge doivent être transférées sont critiques, et donc l'information de charge doit être précise et à jour.

Si la décision d'´equilibrage de charge est faite avant l'exécution de l'application et en connaissant de toutes les variables qui peuvent affecter cet exécution, on parle d'un équilibrage de charge statique. Mais, si on ne connaît pas toutes les variables qui peuvent affecter l'exécution et donc les décisions d'´equilibrage doivent être faites pendant l'exécution, on parle d'un équilibrage de charge dynamique. Notre recherche se focalise dans l'équilibrage de charge dynamique.

Dans l'équilibrage de charge dynamique, les décisions dépendent de l'information rassemblée du système. L'information de charge peut être mise en commun entre les processeurs périodiquement ou sur la demande, avec des collecteurs centralisés ou distribués de l'information. Les algorithmes d'équilibrage de charge se concentrent sur la stabilité (capacité d'´equilibrer la charge seulement si cette action améliore l'exécution du système) et le temps de réponse (capacités de réaction par rapport à des instabilités). Le travail de Casavant et de Kuhl reporte qu'un temps de réponse plus rapide est plus important que la stabilité pour améliorer l'exécution.

Typiquement, un algorithme d'équilibrage de charge a un indice de charge et un ensemble de politiques basées sur l'index. Généralement, les politiques peuvent être classifiées dans une des catégories suivantes. Une politique de partage, définit quelle information doit être partagée et comment elle doit être rassemblée et partagée. Une politique de transfert, détermine quel travail doit être équilibrée et quand le faire. Finalement, une politique de localisation qui détermine où le travail doit être partagé. Ils existent deux genres de politiques de localisation: migration et placement. La migration du travail est réalisée dans le temps d'exécution et le placement est le premier placement d'une application parallèle.

4.1 Indice de charge

Le meilleur indice de charge doit prédire la performance d'une tache exécutée à un nœud particulier (bonne corrélation avec le tempe de réponse d'une tache), aussi aidé à prédire la charge dans un future période étant donné que le temps de réponse d une tache sera plus influencée par le futur change que par la charge actuelle, Etre relativement stable. Notez que ce point est influencée par l'indice de charge et la périodicité de la mesure de charge et Relativement pas cher pour calculer.

Plusieurs indices de charge ont été proposés :

- cpu queue length
- I/0 queue length
- use d Memory
- cpu utilisation

La longueur de la file d attente de cpu est plus adéquaté comme indice de charge car il détermine le comportement de la machine et pas cher pour calculer et relativement stable.

Les systèmes d'équilibrage de charge peuvent être présents en termes de politiques et de mécanismes. Les politiques sont un ensemble de choix conceptuels établis pour déterminer les objectifs du système de distribution de la charge. Les mécanismes réalisent la distribution physique de la charge et fournissent toute information requise par les politiques. La Figure illustre la décomposition d'un système d'équilibrage de charge en six éléments distincts : trois politiques de décision devant être prises (participation, sélection de la localisation et sélection des candidats), et trois principaux mécanismes devant être crées pour supporter les politiques (transfert de la charge de travail, métrique de la charge et communication de la charge).

Nous décrivons dans ce qui suit ces composants [19] [18].

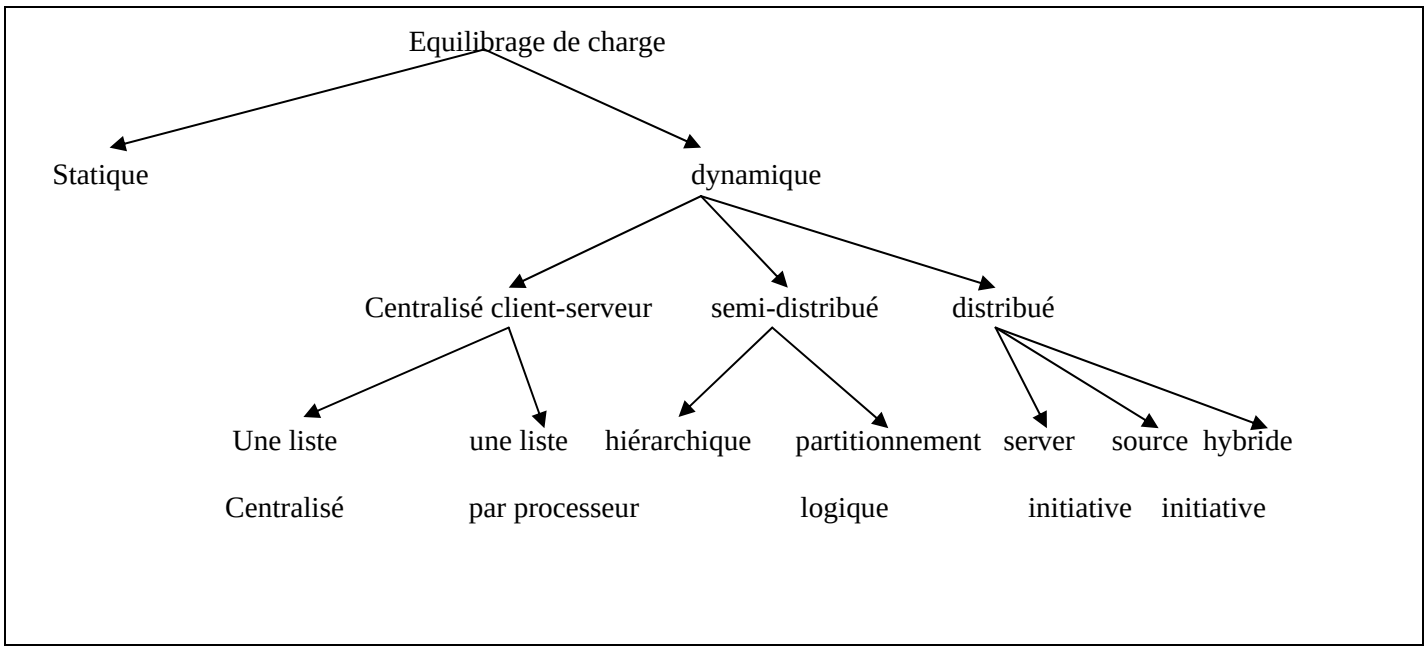

Figure.2: classification des méthodes d'équilibrage de charge dynamique

4.2 Politiques d'équilibrage de charge

4.1.1 Politique de participation

La politique de participation détermine si une ressource participe ou non a l'équilibrage de charge. Les critères de participation peuvent varies. Nous citons dans ce qui suit, deux critères qui sont les plus communs : politique de seuil et politique de possession [19].

Politique de seuil :

Elle détermine si une ressource est sous-charge ou surchargée, selon que sa charge actuelle est en dessous ou en dessus du seuil. La forme la plus commune de cette politique est la considération de deux seuils : le premier définit un niveau inferieur en dessous duquel la ressource est considérée comme receveur, et le deuxième définit un niveau supérieur a partir du quel la ressource est considérée comme source.

Politique de possession :

La possession d'une ressource est considérée, dans certains systèmes, comme la base de la politique de participation. La ressource accepte les taches des autres ressources seulement si elle est inactive. D'autre part, elle est toujours préparée a transférer des taches aux ressources inactives, et a révoquer sa participation et rejeter les taches étrangères si une tache propriétaire est activée.

4.1.2 Politique de transfert, sélection et localisation

Un algorithme d'´ équilibrage de charge a un index de charge et un ensemble de politiques basées sur l'index. Généralement, les politiques peuvent être classifiées dans une de catégories suivantes. Une politique de partage, définit quelle information doit être partagée et comment elle doit être rassemblée et partagée. Une politique de transfert, détermine quel travail doit être équilibrée et quand le faire. Finalement, une politique de localisation qui détermine ou le travail doit être partagé. Il 'existent deux genres de politiques de localisation: migration et placement. La migration du travail est réalisée dans le temps d'exécution et le placement est le premier placement d'une application parallèle. Autant que dans cette thèse nous nous focalisons dans la politique migratoire, nous verrons tout au long de ce travail que le premier placement est une question clé dans l'équilibrage de charge des objets actifs.

- **Politique de transfert** : quand un nœud devienne Sender. La politique de transfert détermine si un nœud est dans un état approprié pour participer à un transfert de tâche comme source ou comme receveur. La politique de transfert peut être basée sur des seuils ou être relative.

La politique de seuil décide qu'un nœud est source, si son indice de charge excède un seuil supérieur Tsé, ou receveur, si son indice de charge est au-dessous d'un seuil inférieur Ti. Le choix de ces seuils est fondamental pour la performance de l'algorithme de distribution de charge. Évidemment, les meilleures valeurs de ces seuils dépendent de la charge du système et du coût de transfert d'une tâche. Lorsque la charge est basse ou lorsque les coûts de transfert sont bas, les seuils doivent favoriser le transfert des tâches. Par contre, lorsque la charge est élevée ou lorsque les coûts de transfert sont élevés, l'exécution à distance doit être évitée. Eager et al. Affirment que le seuil optimal n'est pas très sensible à la charge du système. Ont présenté des techniques qui adaptent dynamiquement et de manière efficace le seuil à la charge du système.

La politique de transfert relative Considère la différence entre la charge d'un nœud et les charges des autres nœuds dans son domaine. Les nœuds sont considérés capables de participer à un transfert si leurs charges diffèrent d'une valeur supérieure à un certain seuil. Ils peuvent alors transférer un nombre fixe de

tâches ou une fraction de la différence de charge. La politique de transfert peut être périodique ou déclenchée par événement. Cependant, les politiques proposées dans la littérature sont dans la grande majorité des cas des politiques déclenchées par événement (fin de traitement d'une tâche ou arrivée d'une tâche).

- **Politique de sélection** : comment Sender choisi job pour le transfert. Une fois que la politique de transfert décide qu'un nœud est source (surchargé), alors la politique de sélection est responsable du choix des tâches à transférer. Nous distinguons quatre méthodes principales de sélection des tâches candidates au transfert :

Choisir une des tâches ayant contribué à ce que le nœud devienne surchargé, choisir n'importe quelle tâche

– Toutes les tâches sont considérées comme éligibles pour la distribution de charge. Aucun filtrage des tâches n'est effectué, choisir une tâche raisonnable

– Le filtrage des tâches est effectué pour éliminer celles qui ne conviennent pas au transfert.

Ces tâches peuvent être rejetées de deux façons :

- statiquement : par analyse des traces du système.
- dynamiquement : un enregistrement dynamique du comportement des tâches peut être maintenu, et les tâches ayant mal répondu à l'exécution distante peuvent être évitées dans le futur.

Choisir une tâche appropriée

– Le choix d'une tâche appropriée peut nécessiter une bonne connaissance sur la tâche aussi bien que sur les machines destinataires. Les systèmes utilisant la migration peuvent obtenir l'information sur le comportement de la tâche courante au moyen de la surveillance de chaque tâche lors de son exécution. La sélection des bons candidats peut être effectuée en se référant à ce comportement. Pour les systèmes effectuant le placement initial, les bons candidats peuvent être sélectionnés à l'aide de la prédiction et de la classification.

- **politique de localisation :** quel nœud devrait être le nœud receveur. Cette politique est responsable de trouver pour un nœud donné un partenaire convenable (source ou receveur) une fois que la politique de transfert a décidé que ce nœud est source ou receveur. Dans une politique centralisée, pour trouver un partenaire convenable pour la distribution de charge, un nœud doit s'adresser au coordinateur qui collecte l'information sur le système (ceci est la tâche de la politique d'information). La politique de transfert utilise ces informations pour choisir des nœuds sources ou receveurs. Une politique de localisation distribuée largement utilisée emploie l'interrogation (polling) pour trouver un nœud convenable. C'est-à-dire qu'un nœud interroge un

autre nœud pour déterminer s'il est convenable ou non. Les nœuds peuvent être interrogés aussi bien en séquentiel qu'en parallèle (multicast). Une alternative à l'interrogation est la diffusion (broadcast), dans laquelle une requête est diffusée aux autres nœuds dans le but de rechercher un nœud disponible pour la distribution de charge.

Certaines politiques de localisation utilisent des modèles probabilistes au lieu de modèles basés sur l'état des nœuds. Ces modèles probabilistes distribuent les tâches selon un ensemble de règles prédéfinies (vecteurs de probabilités).

Des études ont montré que les politiques de localisation basées sur l'état du système offrent de meilleurs résultats que leurs contreparties probabilistes.

La politique de localisation peut prendre en considération certaines restrictions lors de la recherche d'un nœud partenaire. Ces restrictions peuvent inclure : les besoins en ressources, la précédence des tâches, et l'affinité aux données. L'affinité aux données signifie que les tâches peuvent avoir besoin de certaines données qui doivent être extraites à partir de nœuds distants.

5 Mécanismes d'équilibrage de charge

5.1 Mécanisme de transfert

Le mécanisme de transfert représente le protocole employé pour transférer les taches entre les nœuds [19]. Le transfert peut être envisage en deux phases : les requêtes sont allouées a la ressource avant de commencer leur exécution ou âpres l'avoir commence. Ces deux phases sont appelées respectivement placement initial ou processus de migration.

- Mécanisme de mesure de la charge

Le terme charge décrit le niveau d'utilisation des ressources d'un système, et n'est pas une quantité mesurable directement. Le but de la méthode de mesure de la charge est de se focaliser sur un ensemble de ressources qui sont de bons indicateurs de charge pour une certaine politique et qui permettent de comparer différentes ressources. Quelques ressources peuvent être mesurées telle que la mémoire, par contre, d'autres nécessitent d'être déduites, par 34 exemple, l'utilisation CPU est estimée en examinant la taille de la file d'attente.

5.2 Mécanisme de communication de la charge

Le mécanisme de communication de la charge détermine la méthode par la quelle l'information de charge est communiquée entre la ressource et les agents responsables de la prise de décision. Cependant, ce mécanisme pose des problèmes de sécurité, de performance et du cout implique par la collecte et la distribution de l'information de charge. Des solutions a ces problèmes ont été proposées et peuvent être résumées dans les méthodes suivantes

- Polling : le polling est un message dirige a une seule ressource pour retourner la charge actuelle. Il est accompli sur demande, résulte le plus souvent a d'informations de mise a jour, mais a un cout potentiellement élève.
- Broadcasting : le broadcasta est une forme de communication indirecte dans laquelle toutes les ressources s'échangent des informations par diffusion sur le réseau. Cependant, un certain nombre de problèmes peuvent survenir de cette approche : - Elle peut causer un énorme trafic sur le réseau. - Toutes les ressources reçoivent les mises à jour de la charge même si elles ne sont pas concernées par la migration.
- Multicasting : le multicast est une forme de broadcasta dans laquelle, seuls les membres du groupe sont concernes. Elle résout ainsi, un des problèmes poses par le broadcaste, puisque seuls les membres du groupe reçoivent l'information de charge. Cependant, le problème du trafic reste pose.
- Agent Collecteur : la méthode de l'agent collecteur utilise un seul point de collecte de l'information plutôt qu'un algorithme distribue. Trois variantes de cette méthode peuvent

être identifiées, et qui différent dans la manière a laquelle l'information de charge est collectée et dans le moment ou elle est distribuée [20].

- Global : toutes les ressources transmettent périodiquement leurs charges a l'agent central de collection. Si la charge d'une ressource n'a pas change depuis la dernière période, alors aucune mise a jour n'est envoyée a l'agent de collecte. Le vecteur de charge accumule est diffuse périodiquement par l'agent central de collecte a toutes les ressources du système.
- Central : le système possède également, un agent central, seulement, la charge n'est pas collectée ou distribuée périodiquement. Si une ressource veut transférer une tache, elle envoie une requête ainsi que sa charge a l'agent central. Ce dernier répond par une destination suggérée. L'information de charge maintenue par l'agent central est mise a jour seulement par des requêtes de distribution.
- Centex : cette approche combine les deux précédentes et dans laquelle, l'agent central collecte toutes les mises à jour de la charge qui sont effectuées périodiquement, seulement, il ne les distribue pas automatiquement. Si une ressource a besoin de localiser une destination alors elle contacte l'agent central pour avoir l'information la plus récente.

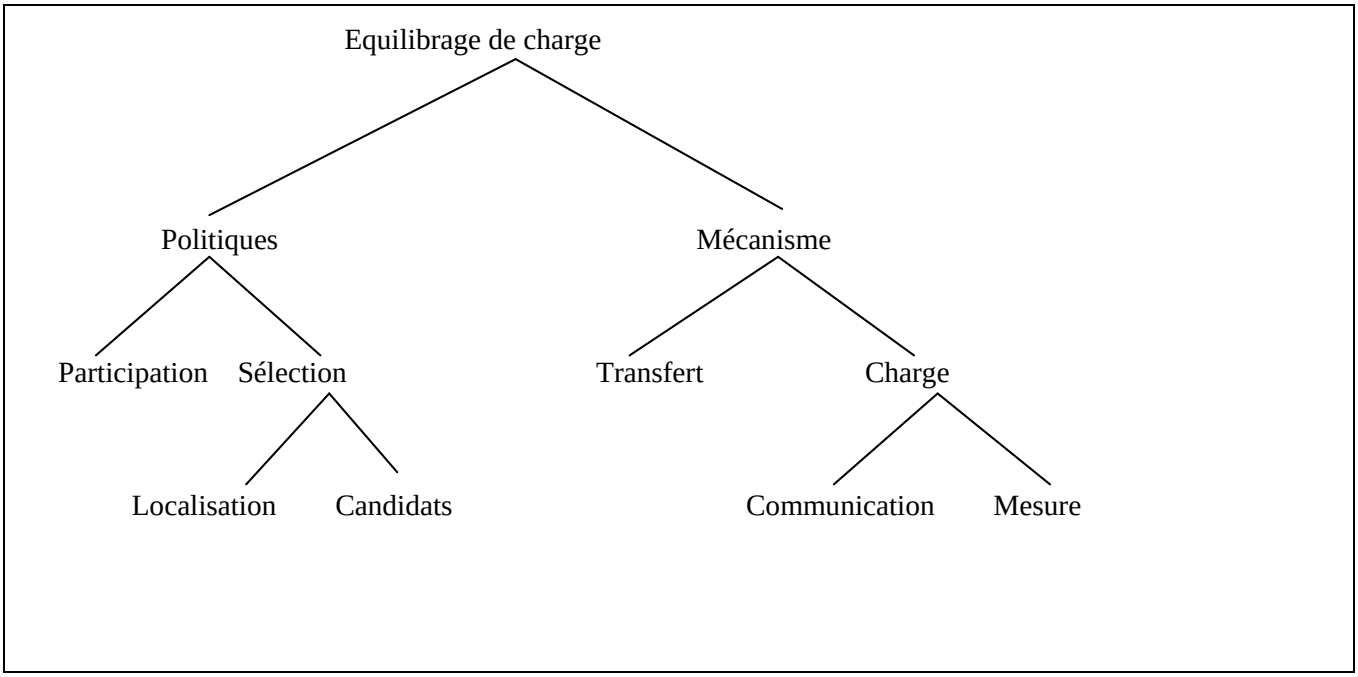

Figure.3 : Composantes d'un système d'équilibrage de charge

6 Algorithmes d'équilibrage

Nous définissons 3 niveaux d'algorithmes : *Intra-grappe*, *Extra-grappe* et *Inter-grappes*.

6.1 Algorithme d'équilibrage Intra-grappe

Cet algorithme constitue le noyau de notre stratégie. L'approche de localité adoptée fait que c'est le niveau d'équilibrage qui sera le plus fréquemment sollicité. Il est déclenché lorsqu'un gestionnaire d'EC's constate qu'il y a déséquilibre entre ses EC's. Pour faire ce constat, le gestionnaire reçoit, de manière Périodique, les informations de charge à partir de chaque élément de calcul. Sur la basede ces informations et du seuil d'équilibrage estimé ", il analyse de manière régulière la charge du groupe. En fonction du résultat de cette analyse, soit il décide de déclencher un équilibrage local en cas de déséquilibre, soit il décide d'informer son gestionnaire du niveau supérieur sur sa charge actuelle.

6.2 Algorithme d'équilibrage Extra-grappe

Cet algorithme, qui utilise une approche *source-initiative*, est exécuté uniquement lorsque certains gestionnaires d'EC's n'ont pas réussi à équilibrer localement leur charge pour cause de saturation ou d'offre insuffisante.

Dans ce cas, le gestionnaire de grappes tente d'équilibrer la charge globale de l'extra-grappe à travers les éléments qu'il gère. Contrairement à l'algorithme *intra grappe*,

L'équilibrage extra-grappe devra tenir compte des coûts de communication.

Durant le transfert de tâches, nous choisirons comme grappe réceptrice, celle qui nécessite le plus petit coût de transfert. Ainsi, le critère de sélection sera pondéré par le coût de communication (coût de transfert de tâches) pour assurer qu'une tâche ne peut être transférée que lorsque son temps de réponse estimé dans la grappe réceptrice, auquel nous rajoutons le temps de transfert à partir de la grappe source, est meilleur que son temps de répons dans la grappe source.

6.3 Algorithme d'équilibrage Inter-grappes

Ce troisième niveau d'algorithme procède un équilibrage global à travers toutes les extra-grappes de la grille. Il est exécuté dans le cas extrême où la majorité des gestionnaires de grappes n'arrivent pas à équilibrer localement leur surplus de charge. Le recours à ce type d'équilibrage ne sera effectif que si les coûts de communication, associés aux différentes tâches à transférer, seront réellement intéressants. En d'autres termes, il serait plus judicieux de laisser un ensemble détaches attendre la libération de ressources que de procéder à leur transfert vers d'autres sites sans un gain de temps substantiel.

Dans ce qui suit, nous allons décrire l'algorithme générique associé à notre stratégie.

7 Les algorithmes source initiative

Les algorithmes source initiative délèguent la gestion de l'équilibre de charge aux processeurs surchargés [GK 94, EB 93, CG 94, BS 85]. Eager et al. proposent trois algorithmes de ce type afin de les comparer suivant le niveau d'information qu'ils utilisent. Les auteurs font deux hypothèses sur l'homogénéité du système:

- tous les processeurs sont identiques ;

- tous les processeurs sont soumis à la même loi d'arrivée des tâches.

7.1 Une stratégie source initiative aléatoire

C'est la technique la plus simple.

Elle ne nécessite aucune information sur l'état des autres processeurs.

Quand un processeur est surchargé, il choisit aléatoirement un processeur à qui il transmet une tâche. Un processeur qui reçoit une tâche peut adopter plusieurs attitudes. En effet, si un processeur qui reçoit une charge décide de la transférer à nouveau car il est lui-même surchargé, le système risque d'avoir un comportement instable. Au contraire, si ce processeur choisit de conserver malgré tout cette tâche, il risque d'être fortement pénalisé. Afin d'éviter ces deux cas extrêmes, les auteurs autorisent le transfert d'une tâche un nombre limité de fois. D'après les auteurs, cette méthode très simple, permet d'améliorer les performances du système.

7.2 Une stratégie source initiative à seuil

Comme pour la méthode précédente, quant un processor. p, est surchargé, il choisit aléatoirement un processeur pi. Avant clé transférer une tâche, le processeur P; vérifie si ce transfert ne risque pas de surcharger le processeur destination (la charge de P1 est au dessus d 'un certain seuil). quant transfert n'est pas possible, un autre processeur est désigné. Cette méthode est appliquée jusqu'à trouver un processeur dont la charge est en dessous du seuil ou bien le nombre clé tentatives est supérieure à une constante définie par l'utilisateur. L'objectif de cette stratégie est d'éviter de réaliser des transferts de tâches inutiles. Comme pour la stratégie aléatoire, un processeur n'essaie pas de faire le meilleur choix mais simplement un choix qui améliore sa situation de façon locale. Le niveau d'information utilisé par cette méthode est peu élevé.

7.3 Une stratégie source initiative du meilleur choix

Un processeur surchargé interroge un certain nombre clé processeurs et tente de réaliser le meilleur choix parmi ce groupe. La mise en couvre de cette stratégie demande un plus grand nombre d'informations Concernant le système. un processeur surchargé interroge un groupe de processeurs choisis aléatoirement. Il transfert une tâche en direction du processeur le moins chargé si le processeur destination n'est pas déjà surchargé. Les résultats obtenus par simulation montrent

que les trois stratégies améliorent de façon significative les performances globales du système. La stratégie la moins performante est. Comme on pouvait s'y' attendre, la stratégie aléatoire. En effet, plusieurs transferts inutiles sont réalisés par cette stratégie qui n'a aucune information sur l'état global du système. Par contre, la stratégie du meilleur choix améliore très peu les résultats obtenus par la stratégie du seuil. Ainsi, les auteurs concluent du L'on peut développer des stratégies source initiative efficaces qui utilisent un nombre réduit d'informations sur l'état du système.

7.4 Un anneau logique

Les stratégies précédentes ont un inconvénient majeur:

il est possible que des tâches soient transférées inutilement ce qui pénalise les performances globales du système. Afin d'éviter ce défaut, Guyenne et al. Proposent de gérer différemment l'ensemble des processeurs IGSS 92, Spi 941. Il propose un algorithme source initiative qui utilise un anneau logique pour réaliser efficacement un équilibrage dynamique clé la charge. Cet anneau logique est constitué par les processeurs déchargés. Chacun des processeurs qui le constituent connaît deux autres processeurs déchargés de l'anneau qui sont situés à sa gauche et à sa droite. De même, chaque processeur dans un état normal ou surchargé connaît un point d'entrée (un numéro de processeur) sur l'anneau logique des processeurs déchargés

À partir de cette structure, un processeur surchargé peut facilement déplacer une tâche vers un processeur déchargé. Il peut dans le cas présent mettre en deux différentes méthodes afin de choisir ce processeur oisif :

- il peut confier directement une tâche au processeur qui est son point d'entrée sur I 'anneau- il peut interroger une partie des processeurs de l'anneau par l'intermédiaire de son point d'entrée afin de faire un meilleur choix ;

- il peut enfin choisir d'interroger tout l'anneau afin de faire le meilleur choix.

La simulation réalisée par les auteurs montre que cet algorithme est très performant. De plus, pour obtenir une efficacité maximale de cet algorithme, il est préférable d'interroger plusieurs processeurs déchargés avant de transférer une tâche, mais il est inutile d'interroger tous les processeurs qui composent l'anneau. Comme pour les algorithmes proposés par Eager et al., c'est la méthode qui utilise un nombre moyen d'informations concernant le système qui obtient les meilleurs résultats.

8 Les algorithmes serveur initiative

Les algorithmes serveur initiative peuvent être vus comme les algorithmes duaux des algorithmes source initiative. Le fait que la responsabilité de l'équilibre de charge soit confiée aux processeurs surchargés est une critique généralement formulée envers les algorithmes source initiative. En effet, leur état est déjà pénalisant pour le système et ce type de stratégie accroît encore leur charge de travail. C'est pourquoi, un grand nombre de chercheurs ont proposé des stratégies où le contrôle de la charge est laissé sous la responsabilité des processeurs oisifs INXG 85, BS 85, LMR 91, WR 93]. Nous proposons de détailler deux algorithmes particuliers qui mettent en couvre une stratégie serveur initiative.

La première apporte une solution globale au problème de l'équilibre de charge . Par conobre, la seconde méthode apporte une solution locale à ce problème.

8.1 La méthode du gradient

F. C. H. Lin et R. M. Keller proposent une stratégie serveur initiative qui apporte une solution globale au problème de l'équilibre de charge. Les auteurs supposent qu'une hypothèse de voisinage est vérifiée. Ainsi, un processeur p possède plusieurs liens de communication en direction de différents processeurs. Ce groupe de processeurs forme le voisinage de p. Il n'est pas possible pour p, de communiquer directement avec un processeur qui n'appartient pas à son voisinage. Le principe de cet algorithme est de permettre à un processeur oisif de le signaler à ses voisins dans le but de récupérer un travail à effectuer. Cette information se propage de proche en proche, par l'intermédiaire des voisins, en direction de tout le système .i i" problème ne peut être résolu de façon locale. Ftc.2'4 -évaluation de la variable proximité et surface du gradient associée tLIi S6l Les auteurs définissent un seuil minimal et un seuil maximal. Quand un processeur est en dessous du seuil minimal, il est considéré comme étant déchargé, de même un processeur est surchargé si sa charge est au dessus du seuil maximal (voir paragraphe;l.i.Z.f ;.

L'intérêt principal de cette méthode à double seuil est d'éviter.,n changement trop rapide de l'état déchargé vers l'état surchargé et inversement. Chaque processeur mesure la distance (le nombre de processeurs) qui le sépare du processeur déchargé qui lui est le plus proche. Quand un processeur a une charge inférieure Att seuil minimal, il a une proximité de 0. De façon générale, la proximité de p est définie de la façon suivante: proximité(p) - min(proximité(n;)) * 1 où les n; sont les voisins de p. Au clé part, la proximité de tous les processeurs est initialisée au)*o, (la distance maximale séparant deux processeurs). Dans cette configuration, l'équilibre de charge est satisfaisant car chaque processeur est au dessus du seuil minimal. À l'aide de cette variable, une surface", supplée" 'surface d'u gradient, est construite. La hauteur d'un point (qui représente un processeur p) est déterminée par la valeur de proximité(p). Ainsi, la surface a une hauteur nulle en un point si le processeur représenté par ce point est déchargé. De même, si la hauteur de la surface est de d pour

un processeur particulier, cela signifie qu'il n'est pas oisif et qu'il se situe à une distance d d'un processeur déchargé. Quand un processeur P est surchargé, il réalise un transfert d'une de ses tâches en direction d'un de ses voisins s'il existe un processeur oisif dans le système (proximité(p) <Lu^o,). Il choisit le voisin dont la variable proximité est minimale. Ainsi la tâche est conduite de proche en proche vers ce processeur qui est inoccupé (voir la figure 2.4). De point de vue visuel, on a I 'impression que les charges émises par des pro."rr".ris surchargés suivent la pente de la surface du gradient. Cet algorithme peut devenir instable dans certain cas. En effet, si le nombre de processeurs déchargé est réduit, un processeur qui est déchargé risque de se retrouver très rapidement surchargé car tous les processeurs surchargés vont émettre dans sa direction des tâches sans aucune concertation. Afin d'éviter cet inconvénient F. J. Muniz et E. J. Zaluska proposent de modifier légèrement cette stratégie IMZ 95]t. Quand un processeur surchargé P' décide de transférer une charge en direction du processeur pi, il 1ui signale au préalable son intention. Pi est libre d'accepter ou de refuser ce transfert. Cette optimisation permet cl 'éviter le transfert inutile de charge à travers le système. Cepenclani, le coût en communication de la méthode du gradient est non négligeable puisqu'un grand nombre de messages de contrôles est induit pour s'assurer de la

validité des informations du système avant de réaliser un déplacement de tâche.

8.2 Une stratégie locale de type serveur initiative

M. lVillebeek-Lemaire et A. P. Reeves proposent une stratégie locale de type serveur-initiative . Ils justifient ce choix par le fait qu'une politique locale est plus adaptée pour une machine fortement parallèle qu'une stratégie globale. En effet, quand le nombre de processeurs est important, il est très coûteux de maintenir à jour un niveau d'information global. Cependant, nous pouvons nuancer cette affirmation des auteurs par le fait général qu'une stratégie trop locale peut prendre de mauvaises décisions car les informations disponibles peuvent être insuffisantes. Dans le cas précis de l'équilibrage de charge, des tâches risquent d'être déplacées inutilement vers un processeur déjà chargé, alors qu'il peut exister des processeurs déchargés qui ne recevront pas de travail car leurs voisins immédiats ne sont pas assez chargés.

Comme pour l'algorithme du gradient, par hypothèse, chaque processeur possède 1{ voisins. Le but de cette stratégie est de maintenir r-rn équilibre de charge local sur l'ensemble des groupes de voisinage définis. Le principe de l'algorithme est le suivant :La charge moyenne locale (pour chaque groupe de processeurs) est calculée régulièrement : t L ,, , è, \ , r r L, : K +TUo + L/6) avec /; la charge du processeur z. ' 'k=l Chatre processeur k surchargé (lr > tro) calcule sa surcharge: h* : l* - Er(ho = 0 pour les autres) K La surcharge totale du système est: Ho =DOo.Le processeur p peut recevoir Lr- lo .frurglitrupplé-"rrtaires. II répartit sa demande sur l'ensemble de ses voisins: e ti- h' 61, : (8, - Ir) * fi """, d* le nombre de tâches prises par p au processeur Ë. Cet algorithme vise à maintenir un état stable, en ne modifiant la charge que de façon locale. Il est

intéressant de noter que I 'utilisateur n'a pas besoin de définir un seuil pour déterminer si un processeur est surchargé ou non. En effet ce seuil est calculé automatiquement à I 'aide de la variable Eo. L'algorithme peut donc s'adapter localement à I 'état du système. Si un déséquilibre important existe au niveau d'un groupe de voisins, un grand nombre de tâches sera transféré des processeurs surchargés vers les processeurs déchargés. Cependant, il est à noter que le nombre de messages de contrôle nécessaires est très important car chaque processeur doit communiquer régulièrement un indicateur de charge à tous ses voisins. Ainsi, pour réaliser une mise à jour de ces informations au niveau du système, chaque processeur émet K messages en direction de ses différents voisins et reçoit 1{ messages en retour. On peut donc estimer qu'une mise à jour du système nécessite lV x /{ échanges de messages où /V est le nombre de processeurs. Pour que cet algorithme soin efficace, il est donc nécessaire de disposer d'un mécanisme de communication efficace entre processeurs voisins.

9 Conclusion :

Dans ce chapitre, nous avons présenté les mécanismes, les politiques et les algorithmes d' équilibrage de charge ,la première partie de chapitre présente cette problématique dans le cadre des systèmes distribués classiques .après nous avons présenté quatre classifications des approches utilisées, les politiques qui composent un système de distribution de charge, La deuxième partie a décrit les algorithmes d' équilibrage de charge pour simplifie les travaux des poste sous chargé et sur chargé , la troisième partie a exposé les algorithmes Sender-initiated (source-initiative) et reciever –initiated (receveur-initiative) ,

Chapitre 2 Conception

1. Introduction

Un des principaux objectifs de l'informatique distribuée est de partager l'accès aux géographiquement distribué ressources hétérogènes de manière transparente. Dans un tel environnement, applications dont les besoins dépassent les ressources de calcul locales peuvent être exécutée. Par ailleurs, la durée moyenne d'emploi de redressement sera réduite grâce à équilibrage de charge sur plusieurs installations informatiques.
L'informatique distribuée a évolué à partir du réseau de stations de travail (NOW) pour grilles de calcul dans le but de devenir une alternative viable aux coûteux dédié machines parallèles .Cependant, un certain nombre de grandes questions techniques doit être manipulé avant que le plein potentiel de l'informatique distribuée peut être réalisé. Travail efficace ordonnancement est une condition essentielle pour l'utilisation efficace des ressources. Dynamique .l'équilibrage de charge est au cœur d'un ordonnanceur de tâches efficace. Bien que de nombreux de chercheurs ont proposé des algorithmes d'ordonnancement pour les architectures parallèles .le problème de la planification de tâches dans un environnement distribué hétérogène fondamentalement différente.
ce chapitre décrit les fonctions d'un simulateur basé sur C destinée à analyser la performance des algorithmes d'équilibrage de charge de trois, à savoir: sender-initiated, receiver-initiated et la combinaison entre les deux Symmetrically –initiated.

2. Problématique

En cette conception, on vas essayer d'étudie comment «' receiver' et 'sender' fonctionne, et aussi, on a essaye de réglé quelque problème qui s'entraver le fonctionne de l ordinateur et cela pour déprisse le fonctionnement avec des postes par le système de distribution la fonction entre les postes pour réalisé l équilibrage de charge, dans le moins de temps possible et nous pouvons résoudre les problèmes de travail, et de mettre une comparaison entre les deux algorithmes.

3. Les facteurs pour lancer l'algorithme d'équilibrage de charge

Il ya quelques facteurs de prise de discision, comme l'arrivée et l'achèvement de travail, l'arrivée et le retrait de ressource, sur la présence de laquelle l'algorithme d'équilibrage de charge introduit .ces facteurs peuvent être résumés comme :

1. Arrivée de n'importe quel nouveau travail
2. Achèvement d'exécution de n'importe quel travail

3. Arrivée de n’importe quelle ressource

4. retrait de n’import quelle ressource

5. la panne de machine à n’importe quel nœud

6. Le nœud devient surchargé : Quand n'importe quel nœud devient pour une raison

Quelconque alors les algorithmes d’équilibrage de charge commencent.

4. Les algorithmes d'équilibrage de charge proposés

Les algorithmes d'équilibrage de charge proposés sont développés en considérant les caractéristiques principales comme la performance, la sortie et l'utilisation de ressource.

4.1 L'algorithme Sender-initiated

L'algorithme Sender-initiated, comme son nom l'indique, est activé par un expéditeur qui souhaite décharger une partie de son calcul. Cet algorithme facilite la migration de travail à partir d'un nœud lourdement chargé à un nœud légèrement chargé Il ya trois décisions fondamentales qui doivent être faites avant le transfert d'un emploi peut avoir lieu:

- Politique de transfert : Quand un nœud ne devienne l'expéditeur?
- Politique de transfert : Comment l'expéditeur de choisir un emploi pour le transfert?
- Politique de sélection : Quel devrait être le nœud récepteur

- **Description des paramètres utilisés**

Nous décrivons ici une description des différents paramètres intervenant dans l'implémentation du cette algorithme.

Paramètres	Signification
SQ	'Sender queue length' Longueur de la file d'attend d'expéditeur
ST	'Sender threshold' Seuil d'expéditeur

Tableau 1 : Description des paramètres utilisés

```
Début
TQ (job arriver)
        Si  SQ+1>ST
        Demander RQs
       SELECT INF
           Si SQ>RQ
               Ajouter (JOB.RQ)
           FIN SI
       SINON
          Ajouter (JOB.SQ)
      FIN SI
  FIN TQ
FIN
```

Algorithme 1 : algorithme Sender initiated

Dans notre simulation la taille de la file d'attente de cpu est le seul indicateur de la charge .le nœud Sender peut utiliser une politique de transfert et initie l'algorithme d'équilibrage lorsque il détecte que sa longe de file d'attente a dépassé un certain seuil un l'arrive d'une nouvelle tache.

la politique de location nécessite la connaissance de la distribution de charge pour localiser les nœud receveur approprie. Le nœud Sender peut envoyer un message à tous les autres nœuds demandant leur taille de file d'attente

.Apres avoir reçu cette information, le nœud Sender peut sélectionner le nœud avec le plus petite taille de la file d'attente.

Comme un nœud-récepteur à condition que la longueur de la file d'attente de Sender(SQ) est supérieur à la langueur de file d'attente du nœud receiver (RQ) (SQ>RQ)

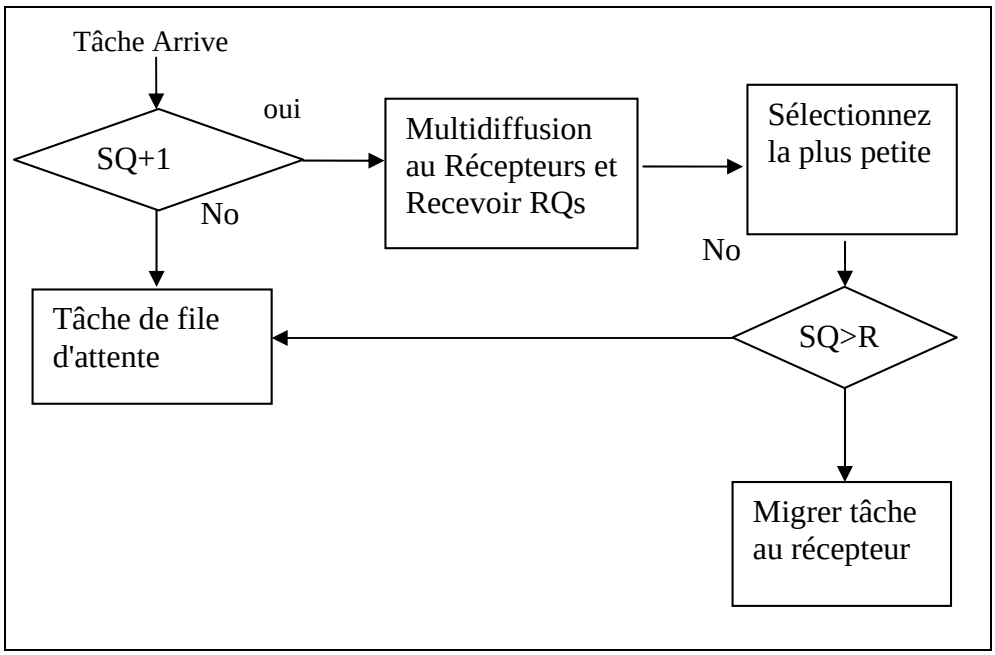

Figure *4 :* l'organigramme de l'algorithme « sender-initiated »

4.2 L'algorithme receiver-initiated

Dans l'algorithme receiver-initiated Un nœud récepteur peut enlever job à partir d'un autre nœud à sa file d'attente.

- Description des paramètres utilisés

Nous décrivons ici une description des différents paramètres intervenant dans l'implémentation du cette algorithme.

Paramètres	Signification
RQ	receiver queue length Longueur de file d'attente de récepteur
RT	Seuil de récepteur

Tableau 2 : Description des paramètres utilisés

```
Début
    TQ (job départ)
    Si  RQ-1<RT
        Demander SQs
        SELECT max
            Si RQ<SQ
                Transfert (JOB.SQ.RQ)
            FIN SI
     SINON
            Exécute la prochaine mission
 FIN SI
     FIN TQ
FIN
```

Algorithme 2 : algorithme reciever initiated

L'algorithme receiver-initiated peut utiliser une politique de transfert similaire de l'algorithme Sender –initiated, tel que il initie l'algorithme d'équilibrage si la longueur de sa file d'attente est inferieurs à un certain seuil(RT) =>RQ<RT Après le départ d'un job. Le même multicast peut être utilisé pour implémenter la politique de location pour identifier le nœud Sender surchargé cependant la politique de sélection nécessite préemption de puis jobs au niveau du nœud Sender ont déjà commencé leur exécution. La décision sur quel travail à enlever n'est pas aussi évidente comme dans l'algorithme Sender initiated .In notre simulation on enlève le dernier job (tâche) dans la file d'attente de nœud Sender.

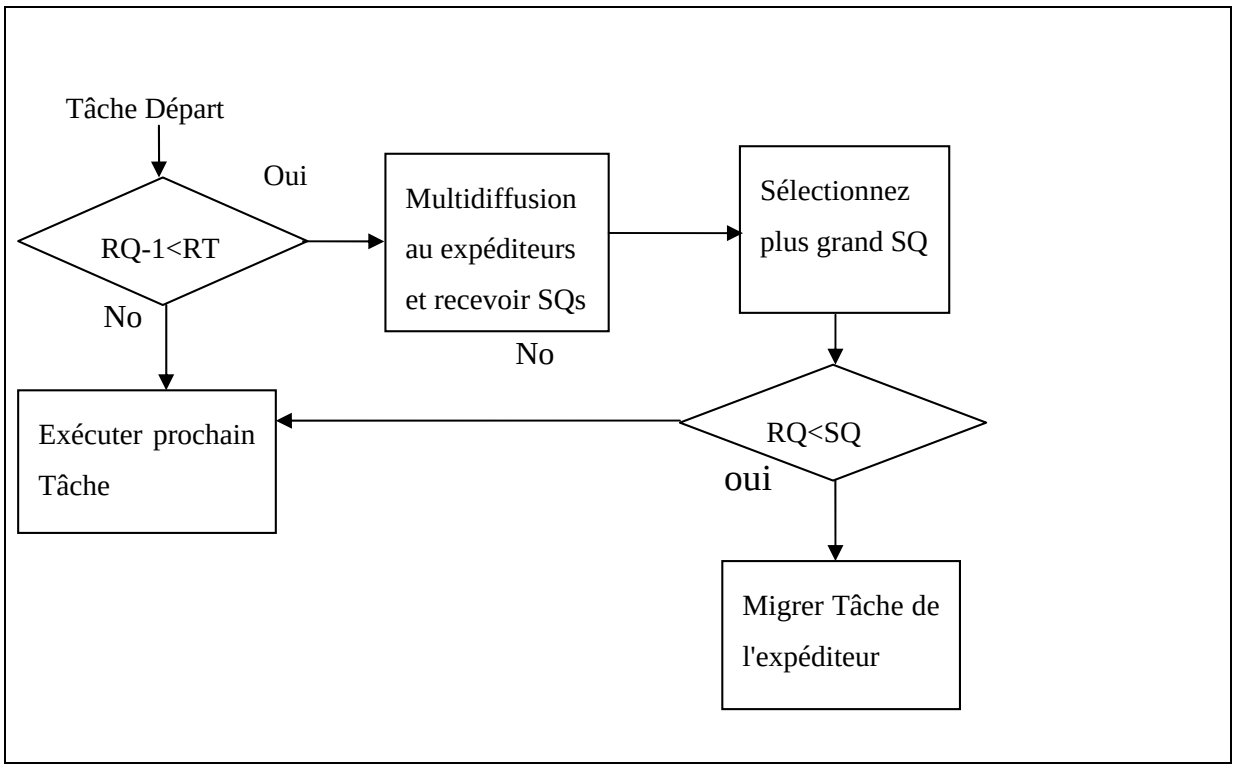

Figure 5: l'organigramme de l'algorithme "Receiver-Initiated"

4.3 L'algorithme Symmetrically-Initiated

Depuis l'expéditeur-initié et récepteur-initié algorithme fonctionne bien à différentes charges du système, il semble logique de les combiner. Un nœud peut activer le Sender-initiated algorithmes lorsque sa taille de file d'attente dépasse un seuil ST, et peut activer le réserver-initiated algorithmes lorsque sa taille de file d'attente descend en dessous de l'autre seuil RT. En tant que tel, chaque nœud peut dynamiquement jouer le rôle soit d'un expéditeur ou un récepteur

5. Conclusion

A travers ce chapitre, nous avons présenté le problématique de l'équilibrage de charge et Les facteurs pour le lancer un algorithme d'équilibrage de charge et nos algorithmes d'équilibrage proposés (sender-initiated, reciever-initiated et Symmetrically-initiated).

Au chapitre suivant, nous présentons les utiles de programmation pour développer nos algorithmes d'équilibrage de charge et les résultats obtenus.

Chapitre 3 Implémentation

1. introduction

Le problème major dans l'implémentation des systèmes dites reparties est l'effet de la surcharge de certains nœuds tandis que d'autres nœud restes en mode repos ou même en état zombie.

Pour remédier ça, il faut implémenter un système ou algorithme qui prend en charge la distribution des ressources entre les nœuds de façon intelligente, et ainsi garder les performances générales hautes et éviter de telles problèmes.

Cette solution dite Le balancement de la charge ou load-balancing, s'identifie par l'implémentation de deux algorithmes différents, plus un autre qui n'est en réalité qu'une implémentation des deux autres de façon combiné.

Dans ce chapitre, nous allons présenter ces trois algorithmes et étudier les performances obtenues en les implémentant, l'une après l'autre en se basant sur un simulateur que nous avons créé pour ces fins-là.

2. L'environnement de développement

Nous avons choisi pour développer notre approche un PC équipé par un processeur pentium 4 CPU 3Ghz, et 2Go de RAM, avec un système d'exploitation XP Pro, et l'outil de développement en "ansi c" (un IDE PellesC come il sera présenté plus tard dans ce chapitre).

Langage «C»

C'est un langage de programmation impératif et généraliste. « C» est qualifié de langage de bas niveau dans le sens où chaque instruction du langage est conçue pour être compilée en un nombre d'instructions machine assez prévisible en termes d'occupation mémoire et de charge de calcul. En outre, il propose un éventail de types entiers et flottants conçus pour pouvoir correspondre directement aux types de donnée supportés par le processeur. Enfin, il fait un usage intensif des calculs d'adresse mémoire avec la notion de pointeur.

Le « C» est un langage compilé (par opposition aux langages interprétés). Cela signifie qu'un programme « C» est décrit par un fichier texte, appelé fichier source. Ce fichier n'étant évidemment pas exécutable par le microprocesseur, il faut le traduire en langage machine. Cette opération est effectuée par un programme appelé compilateur. La compilation se décompose en fait en 4 phases successives :

- **Le traitement par le préprocesseur :** le fichier source est analysé par le préprocesseur qui effectue des transformations purement textuelles (remplacement de chaînes de caractères, inclusion d'autres fichiers source ...).
- **La compilation** : la compilation proprement dite traduit le fichier généré par le préprocesseur en assembleur, c'est-à-dire en une suite d'instructions du microprocesseur qui utilisent des mnémoniques rendant la lecture possible.
- **L'assemblage :** cette opération transforme le code assembleur en un fichier binaire, c'est-à-dire en instructions directement compréhensibles par le processeur. Généralement, la compilation et l'assemblage se font dans la foulée, sauf si l'on spécifie explicitement que l'on veut le code assembleur. Le fichier produit par l'assemblage est appelé fichier objet.
- **L'édition de liens** : un programme est souvent séparé en plusieurs fichiers source, pour des raisons de clarté mais aussi parce qu'il fait généralement appel à des librairies de fonctions standard déjà écrites. Une fois chaque code source assemblé, il faut donc lier entre eux les différents fichiers objets. L'édition de liens produit alors un fichier dit exécutable.

Un programme en langage C est constitué des six groupes de composants élémentaires suivants :

- les identificateurs,
- les mots-clefs,
- les constantes,

- les chaînes de caractères,
- les opérateurs,
- les signes de ponctuation.

On peut ajouter à ces six groupes les commentaires, qui sont enlevés par le préprocesseur.

Le langage C permet d'appréhender des principes informatiques de bas niveau, liés à l'architecture d'un ordinateur, comme la gestion de la mémoire.

Les outils de développement en langage « C »

Pour le développement de cette application, en a choisi un IDE (Integrated Development Environment) qui est un ensemble intégré des outils de compilation, Edition de liens, assemblage …etc. et un éditeur de texte, cette outil dite IDE s'appelle PellesC[1] de "Pelle Orinius", c'est un Outil très léger et très pratique pour les petit projets de ce genre, de plus il est libre et accessible pour tout le monde y compris les professionnels.

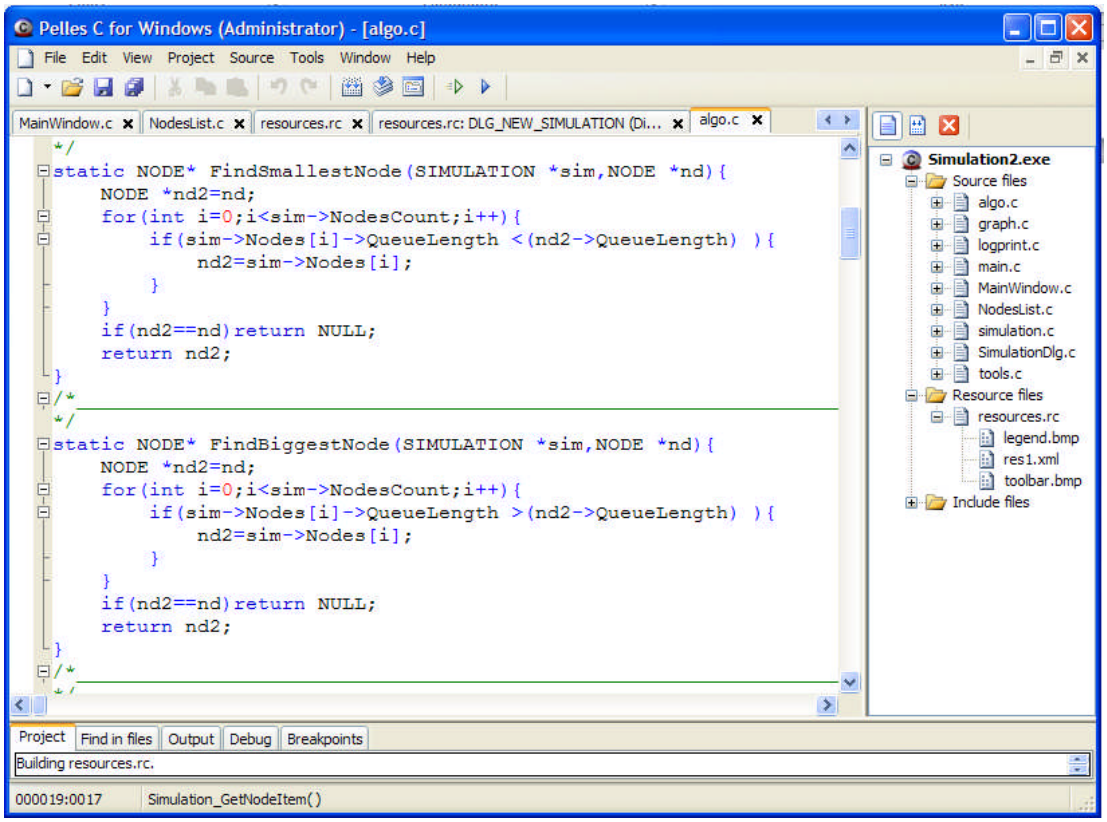

Figure. 6: IDE PellesC - Editeur de code

[1]**Site web PellesC** : www.smorgasbordet.com/**pellesc**

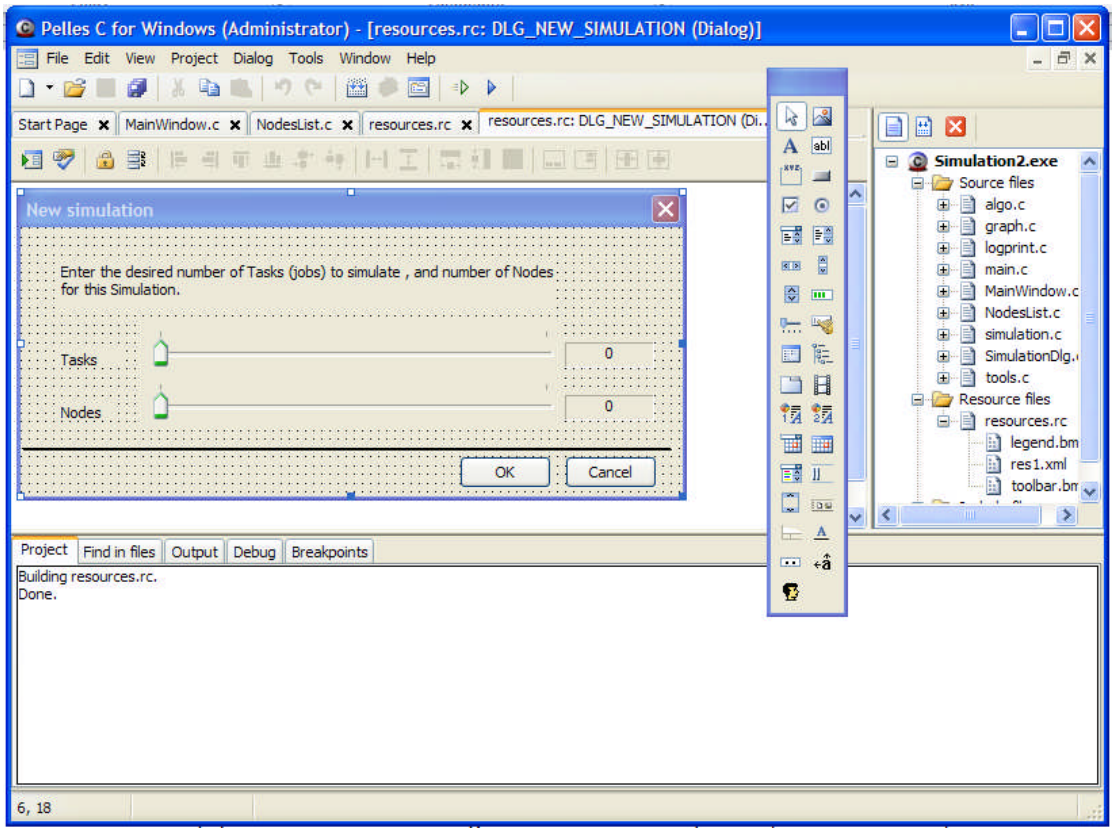

Figure. 7: IDE PellesC - Editeur de ressources

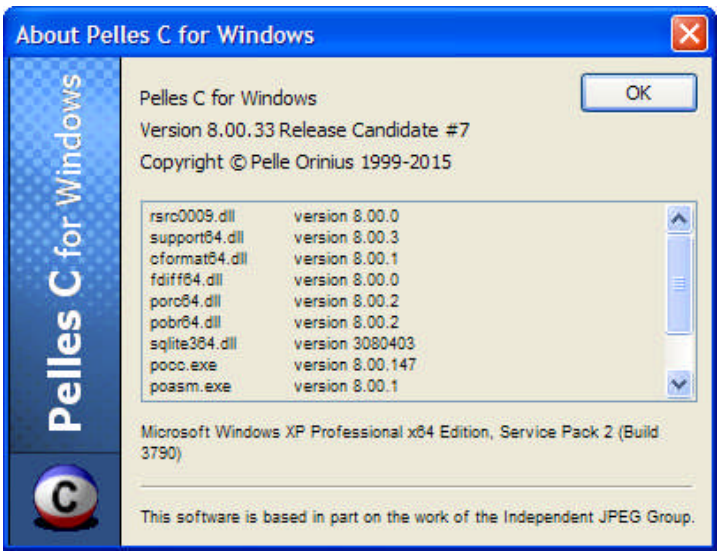

Figure. 8: IDE PellesC : A propos de la version utilisé

3. Le simulateur

Cette section présente le simulateur que nous l'avons développé pour notre projet

L'application a été codée en langage « C » la figure suivante affiche la fenêtre principale du simulateur en action. Le simulateur se divise en trois parties :

- Partie de simulation qui affiche les informations sur les nœuds de simulation.
- Partie de commandes, qui est consistée de boutons de contrôle (Nouvelle simulation, Arrêt, Play et Graph) respectivement. Plus un control Combobox pour le choix de l'algorithme de simulation à savoir (Sender-Initiated,Reciver-Initiatedet Hybrid qui regroupe les deux).
- Et enfin, partie d'affichage d'information générale sur la simulation et son déroulement.

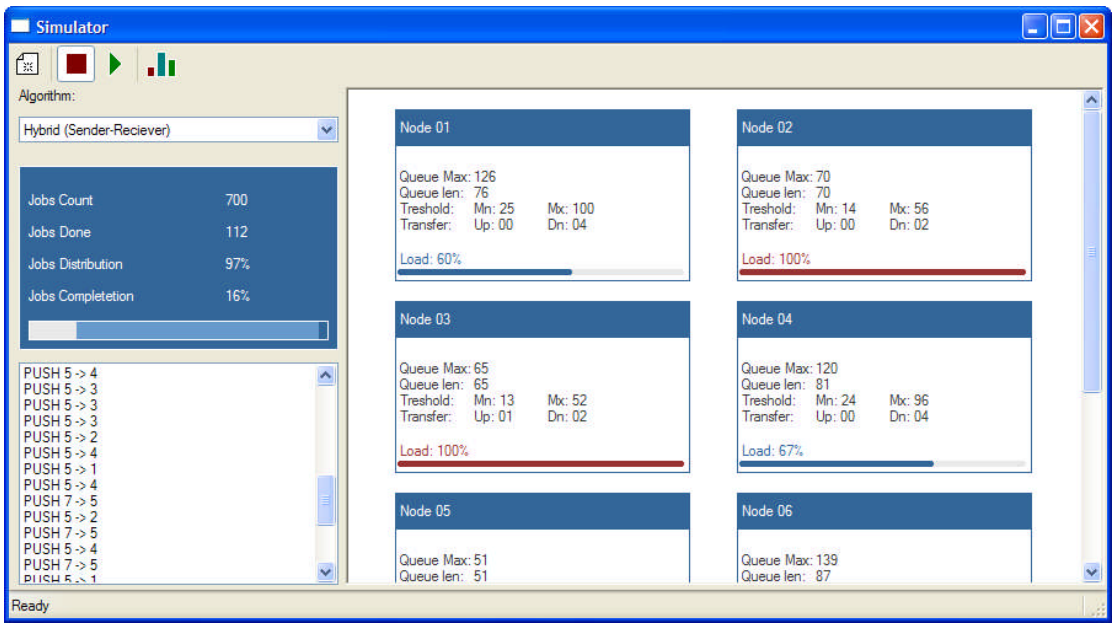

Figure. 9: la fenêtre principale du simulateur

Pour commencer, en click sur le bouton « Nouvelle simulation »  , un boite de dialogue s'affiche pour pouvoir choisir le nombre des Nœuds et celui des taches à exécuter. Les nœuds seront donc créés avec une valeur maximale choisie arbitrairement entre (50 et 150). Les deux limites (**threshold –low et threshold –high**) sont définis respectivement à 20% et 80% de la valeur maximale.

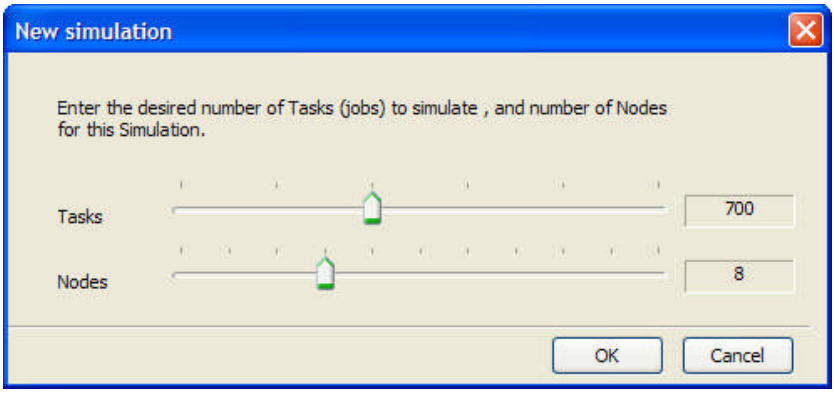

Figure. 10: boite de dialogue "Nouvelle Simulation"

Apres avoir créé une nouvelle simulation, on peut la faire fonctionner selon les différents algorithmes en sélectionnant l'algorithme voulu depuis le contrôle ComboBox (figure suivante) puis appuyer sur le bouton Play.

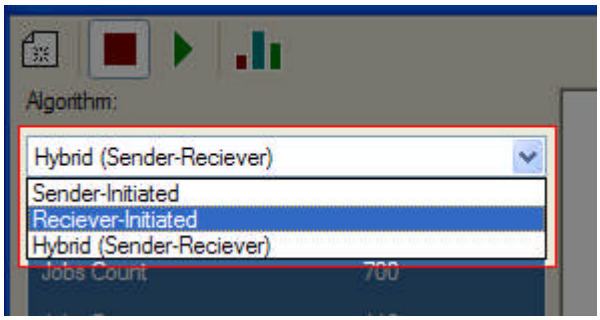

Figure. 11: le control Combobox (Liste des algorithmes)

La simulation commence, la zone d'information affiche la progression de la distribution des taches et le pourcentage d'achèvement de ces taches. La liste en bas quand a elle, elle affiche les différentes actions entre les nœuds eux-mêmes (PUSH et PULL).

4. La simulation de nos algorithmes proposés

L'application met en œuvre la présentation du deux principales algorithmes, le troisième étant la combinaison des deux en même temps.

En fin de simulation une boite de dialogue s'affiche pour présenter le déroulement de la simulation par un graph qui présente la charge totale des nœuds en bleu, les nœuds en repos en vert, et les nœuds en saturation en rouge brique.

Ce qu'en doit s'y attendre est de ne pas avoir des nœuds saturés alors qu'il en existent d'autres en mode repos, et éviter aussi d'avoir des nœuds zombi pendant la simulation.

La bonne optimisation est celle où il n'y a pas trop de nœuds saturés ni trop de nœuds en repos alors que d'autres ne le sont pas.

4.1 L'algorithme Sender-initiated

Cette algorithme comme son nom le dit, est déclenché par le nœud envoyeur, c'est à dire qu'après que le nœud atteint la limite haute du threshold , il commence à chercher parmi les nœuds le moins sollicité, si il en trouve il lui passe la tache active et il continue à exécuter la suivante.

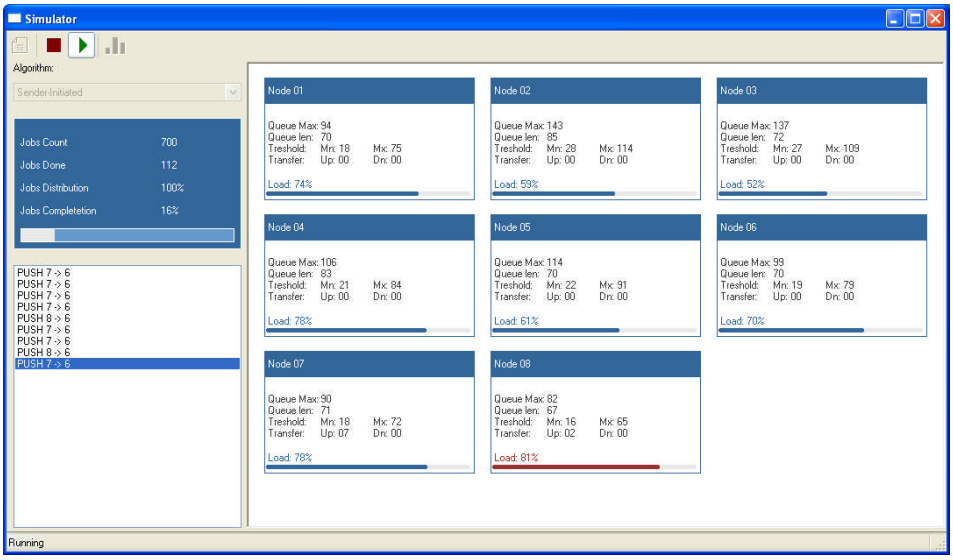

Figure. 12: Sender-initiated en action

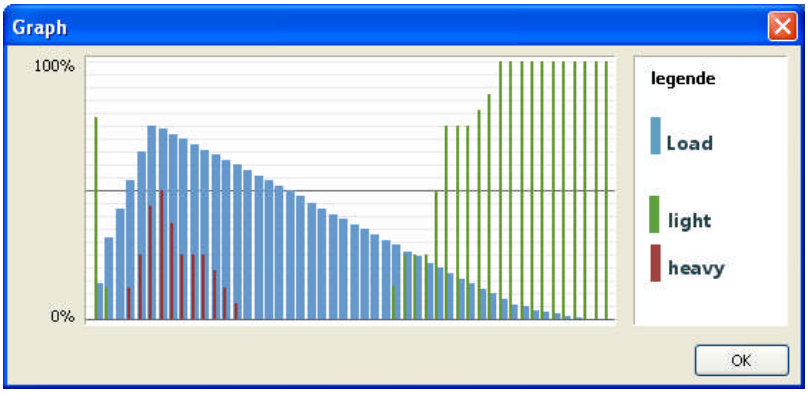

Figure. 13: résultat en Graph de Sender-initiated

Explication des figures:

Dans la figure Figure.12 en vois que le Nœud 8 a atteint le seuil haut (High Threshold) et come le montre la liste des actions à gauche il passe une partie de son travail au Nœud 6.

Dans le graph, en peut voir que les barres en rouge sont moins importantes que celle des verts. Car c'est a ce moment où cette algorithme débute.

4.2 L'algorithme Receiver-Initiated

Cette algorithme contrairement au précèdent, est déclenché par le nœud receveur, c'est à dire qu'après que le nœud atteint la limite basse du threshold, il commence à chercher parmi les nœuds le plus sollicité, si il en trouve un, il exécutera l'une de ces tache, sinon il continuera à exécuter ça propre tache suivante.

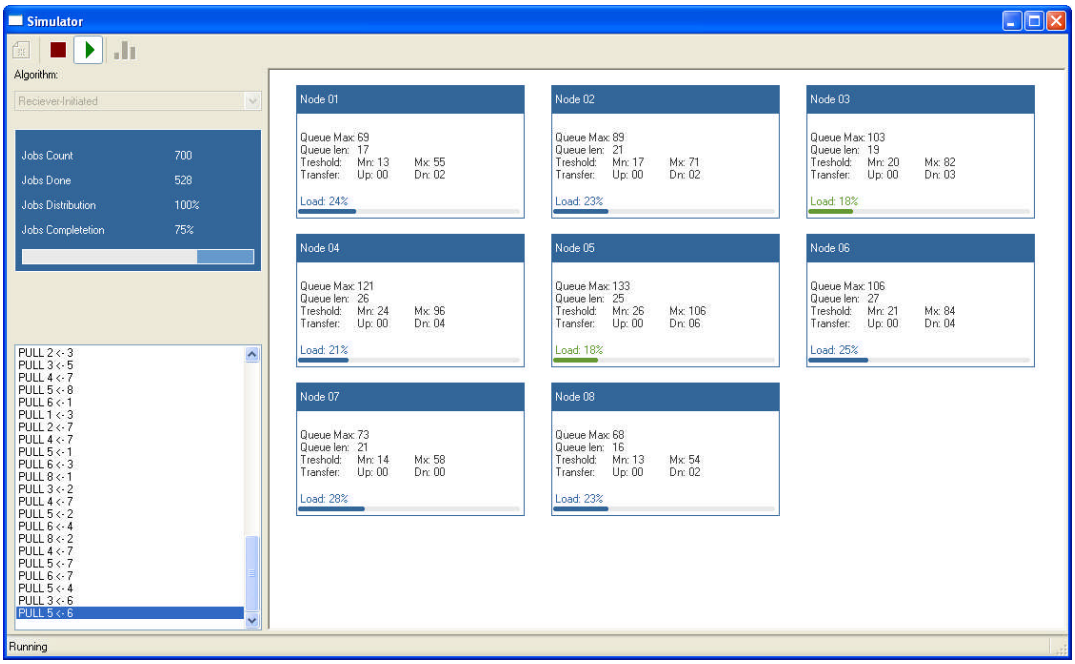

Figure. 14: Receiver-Initiated en action

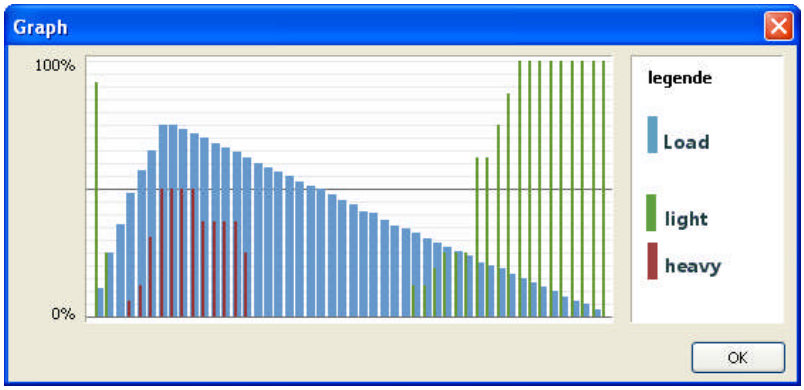

Figure. 1: résultat en graph de Receiver-Initiated

Explication des figures:

La figure Figure.14 montre les Nœuds (3 et 5) en mode PULL, car ils sont en dessous du seuil Minimal (Low Threshold) , la liste à gauche montre bien que ces deux nœuds importe des taches depuis le Nœud N° 6.

Dans Le graph en remarque que les barres rouges ne sont pas optimisés car elles sont plus nombreuse que le graph précèdent. Contrairement aux barres bleues elles le sont. Car cet algorithme ne commence que lorsqu'il est en dessous de la limite minimale (low thershold) là où les nœuds sont en repos.

4.3 L'algorithme Hybrid (Sender &Receiver)

Cet algorithme souvent appelé aussi "symmetrically-initiated algorithm" n'est rien d'autre qu'une combinaison des deux algorithmes précédents. Le but de cet algorithme est d'en tirer profit des performances obtenues par l'utilisation des algorithmes Sender-Initiated et Receiver-Initiated en même temps, ainsi en obtient les meilleures performances.

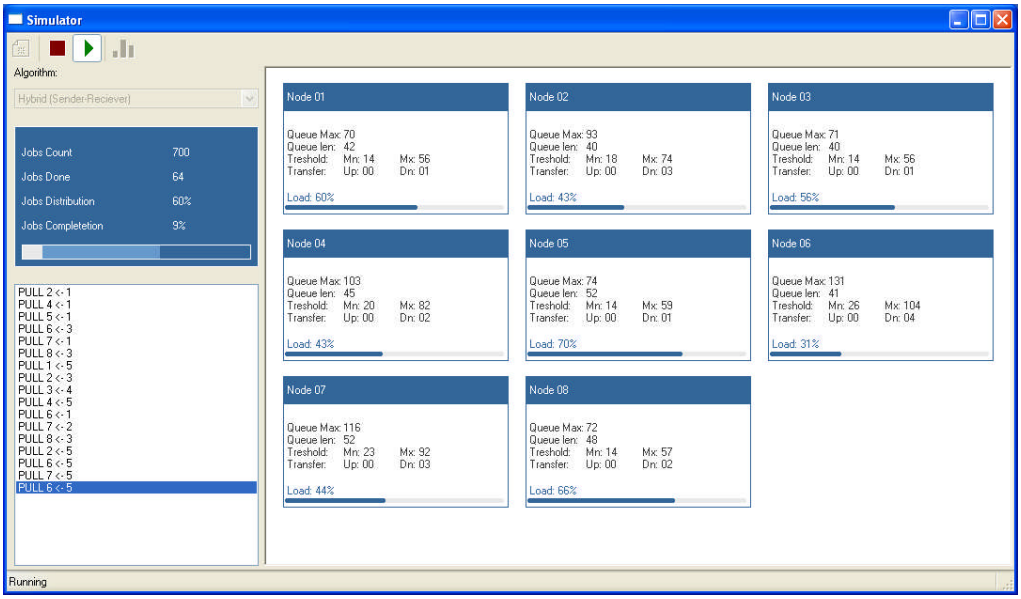

Figure. 16: hybrid en action

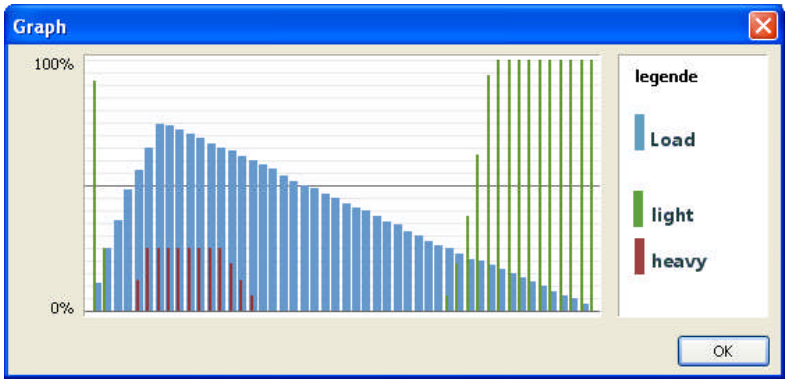

Figure. 2: resultat en graph de l'algorithme hybid

Explication des figures:

La remarque la plus importante dans la figure Figure.16, est que presque tous les Nœuds sont dans le même niveau de chargement, car dans ce mode, la charge est optimisée des le début grâce à l'algorithme (Receiver-Initiated), puis quand les nœuds arrivent à la saturation, l'algorithme (Sender-Initiated) déclenche. Ce qui explique qu'il y a encore moins de nœuds saturés dans ce mode que dans les deux autres, et c'est ce que le graph du figure Figure.17 le montre en claire.

5. Conclusion

Dans ce chapitre, en vient de monter les trois différents algorithmes en action, et montrer de façon pratique en utilisant un programme simulateur créé par nous pour ce but-là. Ainsi, il nous est possible d'étudier par la pratique la différence en question de performances entre l'implémentation de ces algorithmes.

Et comme il était prévu. L'algorithme Hybrid outrepasse les deux autre en matière de performance générale, car il y'a moins de nœuds sollicité que les deux autres et plus de nœuds en repos.

Etant donné que les deux autres implémentent deux différents algorithmes, les performances quand a elles restent presque identiques en générale, le choix d'implémentation entre ces deux algorithmes n'est pas si évident.

Conclusion générale

Les deux stratégies ont un avantage commun par rapport aux méthodes centralisées, élies évitent da formation de goulots d'étranglement. Par contre, pour obtenir un même niveau d'information qu'une stratégie centralisée, le nombre de messages échangés par les méthodes distribuées est plus important. Cependant, il est à noter que des stratégies simples (nécessitant un nombre réduit de messages de contrôle) permettent d'obtenir de bons résultats .

Dans notre recherche nous avons étudier une comparaison entre les deux méthodes (sender et resever) . Nous avons obtenir ces résultats :

- aucune des deux stratégies source et serveur initiative n'est plus efficace dans tous les cas de figures ;

- quand la charge globale du système est faible à moyenne, les stratégies de type source initiative sont plus efficaces que les stratégies serveur initiative ;

- quand la charge globale du système est très élevée, les algorithmes de type serveur initiative sont plus performants que les stratégies source initiative.

Dans le cas d'une machine parallèle dédiée à une seule application, on constate que les méthodes serveur initiative sont très efficaces en début d'exécution et que les stratégies source initiative sont plus performantes en fin d'exécution. En effet, quand l'application débute, il reste encore un grand nombre de tâches à calculer. Le nombre de processeurs déchargés est réduit et il est donc très facile de trouver un processeur surchargé avec qui réaliser un équilibrage de la charge. Par contre, en fin d'exécution, les processeurs surchargés sont rares car l'essentiel des tâches a déjà été calculé. C'est pourquoi, il est préférable que les processeurs surchargés soient à l'origine des nouvelles phases d'équilibre de charge.

Et enfin. L'algorithme Hybrid outrepasse les deux autre en matière de performance générale, car il y'a moins de nœuds sollicité que les deux autres et plus de nœuds en repos.

Bibliography

[1] Michael Mitzenmacher. How useful is old information? IEEE Transactions on Parallel and Distributed Systems, 11(1):6–20, 2000.

[2] Marvin Theimer and Keith Lantz. Finding idle machines in a workstation-based distributed system. IEEE Transactions on Software Engineering, 15(11):1444–1458, 1989.

[3] Thomas Casavant and Jon Kuhl. Effects of response and stability on scheduling in distributed computing systems. IEEE Transactions on Software Engineering, 14(11):1578–1588, 1988.

[4] Thomas Casavant and Jon Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering, 14(2):141–154, 1988.

[5] S.H. Bokhari. A shortest tree algorithm for optimal assignments across space and time in distributed processor system. IEEE transactions on Software Engineering, Vol. SE-7 (6), pp. 583-589, November 1981.

[6] J.V. Hansen & W.C. Giauque. Task Allocation in Distributed processing Systems. Operations Research Letters, Vol 5, N° 3, pp. 137-143, August 1986.

[7] J.B. Sinclair.Efficient computation of optimal assignements for distributed tasks. Journal of Parallel and Distributed Computing, Vol. 4, pp. 342-362, 1987.

[8] W.W. Chu & J-S. Yur. A branch and bound with underestimates algorithm for the task assignement problem with precedence constraint. 10th Int. Conf. on Distributed Computing Systems, Paris, France, pp. 449-501, May 1990.

[9] Jerry C. Yan and Stefen F. Lundstrom. The post-game analysis framework - developing resource management strategies for concurrent systems. IEEE Transactions on Knowledge and Data Engineering, 1(3):293–309, 1989.

[10] Reinhard Riedl and Lutz Richter. Classification of load distribution algorithms. In PDP ’96: Proceedings of the 4th Euromicro Workshop on Parallel and Distributed Processing (PDP ’96), pages 404–413, Washington, DC, USA, 1996. IEEE Com-puter

Society.

[11] Thomas Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. IEEE Transactions on Software Engineering, 17(7):725–730, 1991.

[12] Ravi Mirchandaney, Don Towsley, and John A. Stankovic. Adaptive load sharing in heterogeneous distributed systems. Journal of Parallel and Distributed Computing, 9(4):331–346, 1990.

[14]Talbi. E. G. Une taxonomie des algorithmes d'allocation dynamique de processus dans les systèmes parallèles et distribués . Université de Lille.1997.

[15] Renard .H . Equilibrage de charge être distribution de données sur plates formes hétérogènes . Thèse de Doctorat, Ecole normale supérieure de Lyon, France, Décembre2005.

[16] Badidi .E .Architecture et services pour la distribution de charge dans les systèmes distribués objet. Thèse de PhD, Faculté des études supérieures, Université de Montréal, Mai 2000.

[17] Sagar. D.Load balancing in de laylimited distributed systems. Master's thesis, University of New Mexico Albuquerque, NewMexico, December, 2003.

[18] Yagoubi . B, Hadj Tayeb. L and Si Moussa. H. Load balancing in grid computing . Asian Journal of Information Technology, vol5 (10): pages1095 1103,2006.

[19] Bubendorfer. K and Hine. J. H.A compositional classification for load balancing algorithms. Technical Report, Victoria University of Wellington School of Mathematical and Computing Science, July1998.

[20] Bubendorfer. K. Resource base dpolicies for load distribution. Phd Thesis, Victoria University of Wellington, August 1996.